

NAME

git-diff-tree - Compares the content and mode of blobs found via two tree objects

SYNOPSIS

```
git diff-tree [--stdin] [-m] [-s] [-v] [--no-commit-id] [--pretty]
[-t] [-r] [-c | --cc] [--root] [<common diff options>]
<tree-ish> [<tree-ish>] [<path>...]
```

DESCRIPTION

Compares the content and mode of the blobs found via two tree objects.

If there is only one <tree-ish> given, the commit is compared with its parents (see --stdin below).

Note that *git diff-tree* can use the tree encapsulated in a commit object.

OPTIONS

-p, -u, --patch

Generate patch (see section on generating patches).

-s, --no-patch

Suppress diff output. Useful for commands like *git show* that show the patch by default, or to cancel the effect of --patch.

-U<n>, --unified=<n>

Generate diffs with <n> lines of context instead of the usual three. Implies -p.

--raw

Generate the raw format. This is the default.

--patch-with-raw

Synonym for -p --raw.

--minimal

Spend extra time to make sure the smallest possible diff is produced.

--patience

Generate a diff using the patience diff algorithm.

--histogram

Generate a diff using the histogram diff algorithm.

--diff-algorithm={patience|minimal|histogram|myers}

Choose a diff algorithm. The variants are as follows:

default, myers

The basic greedy diff algorithm. Currently, this is the default.

minimal

Spend extra time to make sure the smallest possible diff is produced.

patience

Use patience diff algorithm when generating patches.

histogram

This algorithm extends the patience algorithm to support low-occurrence common elements.

For instance, if you configured `diff.algorithm` variable to a non-default value and want to use the default one, then you have to use `--diff-algorithm=default` option.

--stat[=<width>[,<name-width>[,<count>]]]

Generate a diffstat. By default, as much space as necessary will be used for the filename part, and the rest for the graph part. Maximum width defaults to terminal width, or 80 columns if not connected to a terminal, and can be overridden by <width>. The width of the filename part can be limited by giving another width <name-width> after a comma. The width of the

graph part can be limited by using `--stat-graph-width=<width>` (affects all commands generating a stat graph) or by setting `diff.statGraphWidth=<width>` (does not affect `git format-patch`). By giving a third parameter `<count>`, you can limit the output to the first `<count>` lines, followed by ... if there are more.

These parameters can also be set individually with `--stat-width=<width>`, `--stat-name-width=<name-width>` and `--stat-count=<count>`.

`--numstat`

Similar to `--stat`, but shows number of added and deleted lines in decimal notation and pathname without abbreviation, to make it more machine friendly. For binary files, outputs two - instead of saying 0 0.

`--shortstat`

Output only the last line of the `--stat` format containing total number of modified files, as well as number of added and deleted lines.

`--dirstat[=<param1,param2,...>]`

Output the distribution of relative amount of changes for each sub-directory. The behavior of `--dirstat` can be customized by passing it a comma separated list of parameters. The defaults are controlled by the `diff.dirstat` configuration variable (see [git-config\(1\)](#)). The following parameters are available:

changes

Compute the `dirstat` numbers by counting the lines that have been removed from the source, or added to the destination. This ignores the amount of pure code movements within a file. In other words, rearranging lines in a file is not counted as much as other changes. This is the default behavior when no parameter is given.

lines

Compute the `dirstat` numbers by doing the regular line-based diff analysis, and summing the removed/added line counts. (For binary files, count 64-byte chunks instead, since binary files have no natural concept of lines). This is a more expensive `--dirstat` behavior than the `changes` behavior, but it does count rearranged lines within a file as much as other changes. The resulting output is consistent with what you get from the other `--*stat` options.

files

Compute the `dirstat` numbers by counting the number of files changed. Each changed file counts equally in the `dirstat` analysis. This is the computationally cheapest `--dirstat` behavior, since it does not have to look at the file contents at all.

cumulative

Count changes in a child directory for the parent directory as well. Note that when using `cumulative`, the sum of the percentages reported may exceed 100%. The default (non-cumulative) behavior can be specified with the `noncumulative` parameter.

<limit>

An integer parameter specifies a cut-off percent (3% by default). Directories contributing less than this percentage of the changes are not shown in the output.

Example: The following will count changed files, while ignoring directories with less than 10% of the total amount of changed files, and accumulating child directory counts in the parent directories: `--dirstat=files,10,cumulative`.

`--summary`

Output a condensed summary of extended header information such as creations, renames and mode changes.

`--patch-with-stat`

Synonym for `-p --stat`.

`-z`

When `--raw`, `--numstat`, `--name-only` or `--name-status` has been given, do not munge pathnames and use NULs as output field terminators.

Without this option, each pathname output will have TAB, LF, double quotes, and backslash characters replaced with `t`, `n`, `,` and `\`, respectively, and the pathname will be enclosed in double quotes if any of those replacements occurred.

`--name-only`

Show only names of changed files.

`--name-status`

Show only names and status of changed files. See the description of the `--diff-filter` option on what the status letters mean.

`--submodule[=<format>]`

Specify how differences in submodules are shown. When `--submodule` or `--submodule=log` is given, the *log* format is used. This format lists the commits in the range like `git-submodule(1)` summary does. Omitting the `--submodule` option or specifying `--submodule=short`, uses the *short* format. This format just shows the names of the commits at the beginning and end of the range. Can be tweaked via the `diff.submodule` configuration variable.

`--color[=<when>]`

Show colored diff. `--color` (i.e. without `=<when>`) is the same as `--color=always`. `<when>` can be one of `always`, `never`, or `auto`.

`--no-color`

Turn off colored diff. It is the same as `--color=never`.

`--word-diff[=<mode>]`

Show a word diff, using the `<mode>` to delimit changed words. By default, words are delimited by whitespace; see `--word-diff-regex` below. The `<mode>` defaults to *plain*, and must be one of:

`color`

Highlight changed words using only colors. Implies `--color`.

`plain`

Show words as `[-removed-]` and `{+added+}`. Makes no attempts to escape the delimiters if they appear in the input, so the output may be ambiguous.

`porcelain`

Use a special line-based format intended for script consumption.

Added/removed/unchanged runs are printed in the usual unified diff format, starting with a `+/-/`` character at the beginning of the line and extending to the end of the line. Newlines in the input are represented by a tilde `~` on a line of its own.

`none`

Disable word diff again.

Note that despite the name of the first mode, `color` is used to highlight the changed parts in all modes if enabled.

`--word-diff-regex=<regex>`

Use `<regex>` to decide what a word is, instead of considering runs of non-whitespace to be a word. Also implies `--word-diff` unless it was already enabled.

Every non-overlapping match of the `<regex>` is considered a word. Anything between these matches is considered whitespace and ignored(!) for the purposes of finding differences. You may want to append `|[[:space:]]` to your regular expression to make sure that it matches all non-whitespace characters. A match that contains a newline is silently truncated(!) at the newline.

The regex can also be set via a diff driver or configuration option, see **gitattributes(1)** or **git-config(1)**. Giving it explicitly overrides any diff driver or configuration setting. Diff drivers override configuration settings.

`--color-words[=<regex>]`

Equivalent to `--word-diff=color` plus (if a regex was specified) `--word-diff-regex=<regex>`.

`--no-renames`

Turn off rename detection, even when the configuration file gives the default to do so.

`--check`

Warn if changes introduce whitespace errors. What are considered whitespace errors is controlled by `core.whitespace` configuration. By default, trailing whitespaces (including lines that solely consist of whitespaces) and a space character that is immediately followed by a tab character inside the initial indent of the line are considered whitespace errors. Exits with non-zero status if problems are found. Not compatible with `--exit-code`.

`--full-index`

Instead of the first handful of characters, show the full pre- and post-image blob object names on the index line when generating patch format output.

`--binary`

In addition to `--full-index`, output a binary diff that can be applied with `git-apply`.

`--abbrev[=<n>]`

Instead of showing the full 40-byte hexadecimal object name in diff-raw format output and diff-tree header lines, show only a partial prefix. This is independent of the `--full-index` option above, which controls the diff-patch output format. Non default number of digits can be specified with `--abbrev=<n>`.

`-B[<n>][/ <m>], --break-rewrites[=[<n>][/ <m>]]`

Break complete rewrite changes into pairs of delete and create. This serves two purposes:

It affects the way a change that amounts to a total rewrite of a file not as a series of deletion and insertion mixed together with a very few lines that happen to match textually as the context, but as a single deletion of everything old followed by a single insertion of everything new, and the number `m` controls this aspect of the `-B` option (defaults to 60%). `-B/70%` specifies that less than 30% of the original should remain in the result for Git to consider it a total rewrite (i.e. otherwise the resulting patch will be a series of deletion and insertion mixed together with context lines).

When used with `-M`, a totally-rewritten file is also considered as the source of a rename (usually `-M` only considers a file that disappeared as the source of a rename), and the number `n` controls this aspect of the `-B` option (defaults to 50%). `-B20%` specifies that a change with addition and deletion compared to 20% or more of the file's size are eligible for being picked up as a possible source of a rename to another file.

`-M[<n>], --find-renames[=<n>]`

Detect renames. If `n` is specified, it is a threshold on the similarity index (i.e. amount of addition/deletions compared to the file's size). For example, `-M90%` means Git should consider a delete/add pair to be a rename if more than 90% of the file hasn't changed. Without a % sign, the number is to be read as a fraction, with a decimal point before it. I.e., `-M5` becomes 0.5, and is thus the same as `-M50%`. Similarly, `-M05` is the same as `-M5%`. To limit detection to exact renames, use `-M100%`. The default similarity index is 50%.

`-C[<n>], --find-copies[=<n>]`

Detect copies as well as renames. See also `--find-copies-harder`. If `n` is specified, it has the same meaning as for `-M[<n>]`.

`--find-copies-harder`

For performance reasons, by default, `-C` option finds copies only if the original file of the copy

was modified in the same changeset. This flag makes the command inspect unmodified files as candidates for the source of copy. This is a very expensive operation for large projects, so use it with caution. Giving more than one `-C` option has the same effect.

`-D, --irreversible-delete`

Omit the preimage for deletes, i.e. print only the header but not the diff between the preimage and `/dev/null`. The resulting patch is not meant to be applied with `patch` or `git apply`; this is solely for people who want to just concentrate on reviewing the text after the change. In addition, the output obviously lack enough information to apply such a patch in reverse, even manually, hence the name of the option.

When used together with `-B`, omit also the preimage in the deletion part of a delete/create pair.

`-l<num>`

The `-M` and `-C` options require $O(n^2)$ processing time where n is the number of potential rename/copy targets. This option prevents rename/copy detection from running if the number of rename/copy targets exceeds the specified number.

`--diff-filter=[(A|C|D|M|R|T|U|X|B)...[*]]`

Select only files that are Added (A), Copied (C), Deleted (D), Modified (M), Renamed (R), have their type (i.e. regular file, symlink, submodule, ...) changed (T), are Unmerged (U), are Unknown (X), or have had their pairing Broken (B). Any combination of the filter characters (including none) can be used. When `*` (All-or-none) is added to the combination, all paths are selected if there is any file that matches other criteria in the comparison; if there is no file that matches other criteria, nothing is selected.

`-S<string>`

Look for differences that change the number of occurrences of the specified string (i.e. addition/deletion) in a file. Intended for the scripter's use.

It is useful when you're looking for an exact block of code (like a struct), and want to know the history of that block since it first came into being: use the feature iteratively to feed the interesting block in the preimage back into `-S`, and keep going until you get the very first version of the block.

`-G<regex>`

Look for differences whose patch text contains added/removed lines that match `<regex>`.

To illustrate the difference between `-S<regex> --pickaxe-regex` and `-G<regex>`, consider a commit with the following diff in the same file:

```
+ return !regexec(regexp, two->ptr, 1, &regmatch, 0);
...
- hit = !regexec(regexp, mf2.ptr, 1, &regmatch, 0);
```

While `git log -Gregexec(regexp)` will show this commit, `git log -Sregexec(regexp --pickaxe-regex)` will not (because the number of occurrences of that string did not change).

See the *pickaxe* entry in [gitdiffcore\(7\)](#) for more information.

`--pickaxe-all`

When `-S` or `-G` finds a change, show all the changes in that changeset, not just the files that contain the change in `<string>`.

`--pickaxe-regex`

Treat the `<string>` given to `-S` as an extended POSIX regular expression to match.

`-O<orderfile>`

Output the patch in the order specified in the `<orderfile>`, which has one shell glob pattern per line. This overrides the `diff.orderfile` configuration variable (see [git-config\(1\)](#)). To cancel `diff.orderfile`, use `-O/dev/null`.

- R
Swap two inputs; that is, show differences from index or on-disk file to tree contents.
- relative[=<path>]
When run from a subdirectory of the project, it can be told to exclude changes outside the directory and show pathnames relative to it with this option. When you are not in a subdirectory (e.g. in a bare repository), you can name which subdirectory to make the output relative to by giving a <path> as an argument.
- a, --text
Treat all files as text.
- ignore-space-at-eol
Ignore changes in whitespace at EOL.
- b, --ignore-space-change
Ignore changes in amount of whitespace. This ignores whitespace at line end, and considers all other sequences of one or more whitespace characters to be equivalent.
- w, --ignore-all-space
Ignore whitespace when comparing lines. This ignores differences even if one line has whitespace where the other line has none.
- ignore-blank-lines
Ignore changes whose lines are all blank.
- inter-hunk-context=<lines>
Show the context between diff hunks, up to the specified number of lines, thereby fusing hunks that are close to each other.
- W, --function-context
Show whole surrounding functions of changes.
- exit-code
Make the program exit with codes similar to [diff\(1\)](#). That is, it exits with 1 if there were differences and 0 means no differences.
- quiet
Disable all output of the program. Implies --exit-code.
- ext-diff
Allow an external diff helper to be executed. If you set an external diff driver with [gitattributes\(5\)](#), you need to use this option with [git-log\(1\)](#) and friends.
- no-ext-diff
Disallow external diff drivers.
- textconv, --no-textconv
Allow (or disallow) external text conversion filters to be run when comparing binary files. See [gitattributes\(5\)](#) for details. Because textconv filters are typically a one-way conversion, the resulting diff is suitable for human consumption, but cannot be applied. For this reason, textconv filters are enabled by default only for [git-diff\(1\)](#) and [git-log\(1\)](#), but not for [git-format-patch\(1\)](#) or diff plumbing commands.
- ignore-submodules[=<when>]
Ignore changes to submodules in the diff generation. <when> can be either none, untracked, dirty or all, which is the default. Using none will consider the submodule modified when it either contains untracked or modified files or its HEAD differs from the commit recorded in the superproject and can be used to override any settings of the *ignore* option in [git-config\(1\)](#) or [gitmodules\(5\)](#). When untracked is used submodules are not considered dirty when they only contain untracked content (but they are still scanned for modified content). Using dirty ignores all changes to the work tree of submodules, only changes to the commits stored in the superproject are shown (this was the behavior until 1.7.0). Using all hides all

changes to submodules.

`--src-prefix=<prefix>`

Show the given source prefix instead of a/.

`--dst-prefix=<prefix>`

Show the given destination prefix instead of b/.

`--no-prefix`

Do not show any source or destination prefix.

For more detailed explanation on these common options, see also [gitdiffcore\(7\)](#).

`<tree-ish>`

The id of a tree object.

`<path>...`

If provided, the results are limited to a subset of files matching one of these prefix strings. i.e., file matches `/^<pattern1>|<pattern2>|.../`. Note that this parameter does not provide any wildcard or regexp features.

`-r`

recurse into sub-trees

`-t`

show tree entry itself as well as subtrees. Implies `-r`.

`--root`

When `--root` is specified the initial commit will be shown as a big creation event. This is equivalent to a diff against the NULL tree.

`--stdin`

When `--stdin` is specified, the command does not take `<tree-ish>` arguments from the command line. Instead, it reads lines containing either two `<tree>`, one `<commit>`, or a list of `<commit>` from its standard input. (Use a single space as separator.)

When two trees are given, it compares the first tree with the second. When a single commit is given, it compares the commit with its parents. The remaining commits, when given, are used as if they are parents of the first commit.

When comparing two trees, the ID of both trees (separated by a space and terminated by a newline) is printed before the difference. When comparing commits, the ID of the first (or only) commit, followed by a newline, is printed.

The following flags further affect the behavior when comparing commits (but not trees).

`-m`

By default, `git diff-tree --stdin` does not show differences for merge commits. With this flag, it shows differences to that commit from all of its parents. See also `-c`.

`-s`

By default, `git diff-tree --stdin` shows differences, either in machine-readable form (without `-p`) or in patch form (with `-p`). This output can be suppressed. It is only useful with `-v` flag.

`-v`

This flag causes `git diff-tree --stdin` to also show the commit message before the differences.

`--pretty[=<format>], --format=<format>`

Pretty-print the contents of the commit logs in a given format, where `<format>` can be one of *oneline*, *short*, *medium*, *full*, *fuller*, *email*, *raw* and *format:<string>*. See the PRETTY FORMATS section for some additional details for each format. When omitted, the format defaults to *medium*.

Note: you can specify the default pretty format in the repository configuration (see [git-config\(1\)](#)).

--abbrev-commit

Instead of showing the full 40-byte hexadecimal commit object name, show only a partial prefix. Non default number of digits can be specified with `--abbrev=<n>` (which also modifies diff output, if it is displayed).

This should make `--pretty=oneline` a whole lot more readable for people using 80-column terminals.

--no-abbrev-commit

Show the full 40-byte hexadecimal commit object name. This negates `--abbrev-commit` and those options which imply it such as `--oneline`. It also overrides the `log.abbrevCommit` variable.

--oneline

This is a shorthand for `--pretty=oneline --abbrev-commit` used together.

--encoding=<encoding>

The commit objects record the encoding used for the log message in their encoding header; this option can be used to tell the command to re-code the commit log message in the encoding preferred by the user. For non plumbing commands this defaults to UTF-8.

--notes[=<ref>]

Show the notes (see [git-notes\(1\)](#)) that annotate the commit, when showing the commit log message. This is the default for `git log`, `git show` and `git whatchanged` commands when there is no `--pretty`, `--format`, or `--oneline` option given on the command line.

By default, the notes shown are from the notes refs listed in the `core.notesRef` and `notes.displayRef` variables (or corresponding environment overrides). See [git-config\(1\)](#) for more details.

With an optional `<ref>` argument, show this notes ref instead of the default notes ref(s). The ref is taken to be in `refs/notes/` if it is not qualified.

Multiple `--notes` options can be combined to control which notes are being displayed.

Examples: `--notes=foo` will show only notes from `refs/notes/foo`; `--notes=foo --notes` will show both notes from `refs/notes/foo` and from the default notes ref(s).

--no-notes

Do not show notes. This negates the above `--notes` option, by resetting the list of notes refs from which notes are shown. Options are parsed in the order given on the command line, so e.g. `--notes --notes=foo --no-notes --notes=bar` will only show notes from `refs/notes/bar`.

--show-notes[=<ref>], --[no-]standard-notes

These options are deprecated. Use the above `--notes/--no-notes` options instead.

--show-signature

Check the validity of a signed commit object by passing the signature to `gpg --verify` and show the output.

--no-commit-id

`git diff-tree` outputs a line with the commit ID when applicable. This flag suppressed the commit ID output.

-c

This flag changes the way a merge commit is displayed (which means it is useful only when the command is given one `<tree-ish>`, or `--stdin`). It shows the differences from each of the parents to the merge result simultaneously instead of showing pairwise diff between a parent and the result one at a time (which is what the `-m` option does). Furthermore, it lists only files which were modified from all parents.

--cc

This flag changes the way a merge commit patch is displayed, in a similar way to the `-c`

option. It implies the *-c* and *-p* options and further compresses the patch output by omitting uninteresting hunks whose the contents in the parents have only two variants and the merge result picks one of them without modification. When all hunks are uninteresting, the commit itself and the commit log message is not shown, just like in any other empty diff case.

--always

Show the commit itself and the commit log message even if the diff itself is empty.

PRETTY FORMATS

If the commit is a merge, and if the pretty-format is not *oneline*, *email* or *raw*, an additional line is inserted before the *Author:* line. This line begins with Merge: and the sha1s of ancestral commits are printed, separated by spaces. Note that the listed commits may not necessarily be the list of the **direct** parent commits if you have limited your view of history: for example, if you are only interested in changes related to a certain directory or file.

There are several built-in formats, and you can define additional formats by setting a pretty.<name> config option to either another format name, or a *format:* string, as described below (see [git-config\(1\)](#)). Here are the details of the built-in formats:

- *oneline*

```
<sha1> <title line>
```

This is designed to be as compact as possible.

- *short*

```
commit <sha1>
Author: <author>
```

```
<title line>
```

- *medium*

```
commit <sha1>
Author: <author>
Date: <author date>
```

```
<title line>
```

```
<full commit message>
```

- *full*

```
commit <sha1>
Author: <author>
Commit: <committer>
```

```
<title line>
```

```
<full commit message>
```

- *fuller*

```
commit <sha1>
Author: <author>
AuthorDate: <author date>
Commit: <committer>
CommitDate: <committer date>
```

```
<title line>
```

```
<full commit message>
```

- *email*

```
From <sha1> <date>
From: <author>
Date: <author date>
Subject: [PATCH] <title line>
```

<full commit message>

- *raw*

The *raw* format shows the entire commit exactly as stored in the commit object. Notably, the SHA-1s are displayed in full, regardless of whether `--abbrev` or `--no-abbrev` are used, and *parents* information show the true parent commits, without taking grafts or history simplification into account.

- *format:<string>*

The *format:<string>* format allows you to specify which information you want to show. It works a little bit like printf format, with the notable exception that you get a newline with `%n` instead of `n`.

E.g, *format:The author of %h was %an, %ar%nThe title was >>%s<<%n* would show something like this:

```
The author of fe6e0ee was Junio C Hamano, 23 hours ago
```

```
The title was >>t4119: test autocomputing -p<n> for traditional diff input.<<
```

The placeholders are:

- `%H`: commit hash
- `%h`: abbreviated commit hash
- `%T`: tree hash
- `%t`: abbreviated tree hash
- `%P`: parent hashes
- `%p`: abbreviated parent hashes
- `%an`: author name
- `%aN`: author name (respecting `.mailmap`, see [git-shortlog\(1\)](#) or [git-blame\(1\)](#))
- `%ae`: author email
- `%aE`: author email (respecting `.mailmap`, see [git-shortlog\(1\)](#) or [git-blame\(1\)](#))
- `%ad`: author date (format respects `--date=` option)
- `%aD`: author date, RFC2822 style
- `%ar`: author date, relative
- `%at`: author date, UNIX timestamp
- `%ai`: author date, ISO 8601 format
- `%cn`: committer name
- `%cN`: committer name (respecting `.mailmap`, see [git-shortlog\(1\)](#) or [git-blame\(1\)](#))
- `%ce`: committer email
- `%cE`: committer email (respecting `.mailmap`, see [git-shortlog\(1\)](#) or [git-blame\(1\)](#))
- `%cd`: committer date
- `%cD`: committer date, RFC2822 style
- `%cr`: committer date, relative
- `%ct`: committer date, UNIX timestamp
- `%ci`: committer date, ISO 8601 format
- `%d`: ref names, like the `--decorate` option of [git-log\(1\)](#)
- `%e`: encoding
- `%s`: subject
- `%f`: sanitized subject line, suitable for a filename
- `%b`: body
- `%B`: raw body (unwrapped subject and body)
- `%N`: commit notes
- `%GG`: raw verification message from GPG for a signed commit
- `%G?`: show G for a Good signature, B for a Bad signature, U for a good, untrusted signature and N for no signature
- `%GS`: show the name of the signer for a signed commit
- `%GK`: show the key used to sign a signed commit
- `%gD`: reflog selector, e.g., `refs/stash@{1}`
- `%gd`: shortened reflog selector, e.g., `stash@{1}`
- `%gn`: reflog identity name
- `%gN`: reflog identity name (respecting `.mailmap`, see [git-shortlog\(1\)](#) or [git-blame\(1\)](#))

- `%ge`: relog identity email
- `%gE`: relog identity email (respecting `.mailmap`, see [git-shortlog\(1\)](#) or [git-blame\(1\)](#))
- `%gs`: relog subject
- `%Cred`: switch color to red
- `%Cgreen`: switch color to green
- `%Cblue`: switch color to blue
- `%Creset`: reset color
- `%C(...)`: color specification, as described in `color.branch.*` config option; adding `auto`, at the beginning will emit color only when colors are enabled for log output (by `color.diff`, `color.ui`, or `--color`, and respecting the auto settings of the former if we are going to a terminal). `auto` alone (i.e. `%C(auto)`) will turn on auto coloring on the next placeholders until the color is switched again.
- `%m`: left, right or boundary mark
- `%n`: newline
- `%%`: a raw `%`
- `%x00`: print a byte from a hex code
- `%w(<w>[,<i1>[,<i2>]]])`: switch line wrapping, like the `-w` option of [git-shortlog\(1\)](#).
- `%<(<N>[,<trunc>[,<mtrunc>]]])`: make the next placeholder take at least `N` columns, padding spaces on the right if necessary. Optionally truncate at the beginning (`ltrunc`), the middle (`mtrunc`) or the end (`trunc`) if the output is longer than `N` columns. Note that truncating only works correctly with `N >= 2`.
- `%<|(<N>)`: make the next placeholder take at least until `N`th columns, padding spaces on the right if necessary
- `%>(<N>)`, `%>|(<N>)`: similar to `%<(<N>)`, `%<|(<N>)` respectively, but padding spaces on the left
- `%>>(<N>)`, `%>>|(<N>)`: similar to `%>(<N>)`, `%>|(<N>)` respectively, except that if the next placeholder takes more spaces than given and there are spaces on its left, use those spaces
- `%><(<N>)`, `%><|(<N>)`: similar to `%<(<N>)`, `%<|(<N>)` respectively, but padding both sides (i.e. the text is centered)

Note

Some placeholders may depend on other options given to the revision traversal engine. For example, the `%g*` relog options will insert an empty string unless we are traversing relog entries (e.g., by `git log -g`). The `%d` placeholder will use the short decoration format if `--decorate` was not already provided on the command line.

If you add a `+` (plus sign) after `%` of a placeholder, a line-feed is inserted immediately before the expansion if and only if the placeholder expands to a non-empty string.

If you add a `-` (minus sign) after `%` of a placeholder, line-feeds that immediately precede the expansion are deleted if and only if the placeholder expands to an empty string.

If you add a `' '` (space) after `%` of a placeholder, a space is inserted immediately before the expansion if and only if the placeholder expands to a non-empty string.

- *tformat*:

The *tformat*: format works exactly like *format*:, except that it provides terminator semantics instead of separator semantics. In other words, each commit has the message terminator character (usually a newline) appended, rather than a separator placed between entries. This means that the final entry of a single-line format will be properly terminated with a new line, just as the oneline format does. For example:

```
$ git log -2 --pretty=format:%h 4da45bef
| perl -pe $_ .= -- NO NEWLINE unless /n/
4da45be
7134973 -- NO NEWLINE

$ git log -2 --pretty=tformat:%h 4da45bef
```

```
| perl -pe $_ .= -- NO NEWLINE unless /n/
4da45be
7134973
```

In addition, any unrecognized string that has a % in it is interpreted as if it has tformat: in front of it. For example, these two are equivalent:

```
$ git log -2 --pretty=tformat:%h 4da45bef
$ git log -2 --pretty=%h 4da45bef
```

LIMITING OUTPUT

If you're only interested in differences in a subset of files, for example some architecture-specific files, you might do:

```
git diff-tree -r <tree-ish> <tree-ish> arch/ia64 include/asm-ia64
```

and it will only show you what changed in those two directories.

Or if you are searching for what changed in just kernel/sched.c, just do

```
git diff-tree -r <tree-ish> <tree-ish> kernel/sched.c
```

and it will ignore all differences to other files.

The pattern is always the prefix, and is matched exactly. There are no wildcards. Even stricter, it has to match a complete path component. I.e. foo does not pick up foobar.h. foo does match foo/bar.h so it can be used to name subdirectories.

An example of normal usage is:

```
torvalds@ppc970:~/git> git diff-tree --abbrev 5319e4
:100664 100664 ac348b... a01513... git-fsck-objects.c
```

which tells you that the last commit changed just one file (it's from this one:

```
commit 3c6f7ca19ad4043e9e72fa94106f352897e651a8
tree 5319e4d609cdd282069cc4dce33c1db559539b03
parent b4e628ea30d5ab3606119d2ea5caeb141d38df7
author Linus Torvalds <torvalds@ppc970.osdl.org> Sat Apr 9 12:02:30 2005
committer Linus Torvalds <torvalds@ppc970.osdl.org> Sat Apr 9 12:02:30 2005
```

Make git-fsck-objects print out all the root commits it finds.

Once I do the reference tracking, Ill also make it print out all the HEAD commits it finds, which is even more interesting.

in case you care).

RAW OUTPUT FORMAT

The raw output format from git-diff-index, git-diff-tree, git-diff-files and git diff --raw are very similar.

These commands all compare two sets of things; what is compared differs:

```
git-diff-index <tree-ish>
```

compares the <tree-ish> and the files on the filesystem.

```
git-diff-index --cached <tree-ish>
```

compares the <tree-ish> and the index.

```
git-diff-tree [-r] <tree-ish-1> <tree-ish-2> [<pattern>...]
```

compares the trees named by the two arguments.

```
git-diff-files [<pattern>...]
```

compares the index and the files on the filesystem.

The `git-diff-tree` command begins its output by printing the hash of what is being compared. After that, all the commands print one output line per changed file.

An output line is formatted this way:

```
in-place edit :100644 100644 bcd1234... 0123456... M file0
copy-edit :100644 100644 abcd123... 1234567... C68 file1 file2
rename-edit :100644 100644 abcd123... 1234567... R86 file1 file3
create :000000 100644 0000000... 1234567... A file4
delete :100644 000000 1234567... 0000000... D file5
unmerged :000000 000000 0000000... 0000000... U file6
```

That is, from the left to the right:

1. a colon.
2. mode for src; 000000 if creation or unmerged.
3. a space.
4. mode for dst; 000000 if deletion or unmerged.
5. a space.
6. sha1 for src; 0{40} if creation or unmerged.
7. a space.
8. sha1 for dst; 0{40} if creation, unmerged or look at work tree.
9. a space.
10. status, followed by optional score number.
11. a tab or a NUL when `-z` option is used.
12. path for src
13. a tab or a NUL when `-z` option is used; only exists for C or R.
14. path for dst; only exists for C or R.
15. an LF or a NUL when `-z` option is used, to terminate the record.

Possible status letters are:

- A: addition of a file
- C: copy of a file into a new one
- D: deletion of a file
- M: modification of the contents or mode of a file
- R: renaming of a file
- T: change in the type of the file
- U: file is unmerged (you must complete the merge before it can be committed)
- X: unknown change type (most probably a bug, please report it)

Status letters C and R are always followed by a score (denoting the percentage of similarity between the source and target of the move or copy), and are the only ones to be so.

<sha1> is shown as all 0's if a file is new on the filesystem and it is out of sync with the index.

Example:

```
:100644 100644 5be4a4..... 000000..... M file.c
```

When `-z` option is not used, TAB, LF, and backslash characters in pathnames are represented as `t`, `n`, and `\`, respectively.

DIFF FORMAT FOR MERGES

`git-diff-tree`, `git-diff-files` and `git-diff --raw` can take `-c` or `--cc` option to generate diff output also for merge commits. The output differs from the format described above in the following way:

1. there is a colon for each parent
2. there are more src modes and src sha1
3. status is concatenated status characters for each parent
4. no optional score number
5. single path, only for dst

Example:

```
::100644 100644 100644 fabadb8... cc95eb0... 4866510... MM describe.c
```

Note that *combined diff* lists only files which were modified from all parents.

GENERATING PATCHES WITH -P

When `git-diff-index`, `git-diff-tree`, or `git-diff-files` are run with a `-p` option, `git diff` without the `--raw` option, or `git log` with the `-p` option, they do not produce the output described above; instead they produce a patch file. You can customize the creation of such patches via the `GIT_EXTERNAL_DIFF` and the `GIT_DIFF_OPTS` environment variables.

What the `-p` option produces is slightly different from the traditional diff format:

1. It is preceded with a git diff header that looks like this:

```
diff --git a/file1 b/file2
```

The `a/` and `b/` filenames are the same unless rename/copy is involved. Especially, even for a creation or a deletion, `/dev/null` is *not* used in place of the `a/` or `b/` filenames.

When rename/copy is involved, `file1` and `file2` show the name of the source file of the rename/copy and the name of the file that rename/copy produces, respectively.

2. It is followed by one or more extended header lines:

```
old mode <mode>
new mode <mode>
deleted file mode <mode>
new file mode <mode>
copy from <path>
copy to <path>
rename from <path>
rename to <path>
similarity index <number>
dissimilarity index <number>
index <hash>..<hash> <mode>
```

File modes are printed as 6-digit octal numbers including the file type and file permission bits.

Path names in extended headers do not include the `a/` and `b/` prefixes.

The similarity index is the percentage of unchanged lines, and the dissimilarity index is the percentage of changed lines. It is a rounded down integer, followed by a percent sign. The similarity index value of 100% is thus reserved for two equal files, while 100% dissimilarity means that no line from the old file made it into the new one.

The index line includes the SHA-1 checksum before and after the change. The `<mode>` is included if the file mode does not change; otherwise, separate lines indicate the old and the new mode.

3. TAB, LF, double quote and backslash characters in pathnames are represented as `t`, `n`, and `,`, respectively. If there is need for such substitution then the whole pathname is put in double quotes.
4. All the `file1` files in the output refer to files before the commit, and all the `file2` files refer to files after the commit. It is incorrect to apply each change to each file sequentially. For example, this patch will swap `a` and `b`:

```
diff --git a/a b/b
rename from a
rename to b
diff --git a/b b/a
rename from b
rename to a
```

COMBINED DIFF FORMAT

Any diff-generating command can take the ‘-c’ or --cc option to produce a *combined diff* when showing a merge. This is the default format when showing merges with [git-diff\(1\)](#) or [git-show\(1\)](#). Note also that you can give the ‘-m’ option to any of these commands to force generation of diffs with individual parents of a merge.

A *combined diff* format looks like this:

```
diff --combined describe.c
index fabadb8,cc95eb0..4866510
--- a/describe.c
+++ b/describe.c
@@@ -98,20 -98,12 +98,20 @@@
return (a_date > b_date) ? -1 : (a_date == b_date) ? 0 : 1;
}
```

```
- static void describe(char *arg)
-static void describe(struct commit *cmit, int last_one)
++static void describe(char *arg, int last_one)
{
+ unsigned char sha1[20];
+ struct commit *cmit;
struct commit_list *list;
static int initialized = 0;
struct commit_name *n;
```

```
+ if (get_sha1(arg, sha1) < 0)
+ usage(describe_usage);
+ cmit = lookup_commit_reference(sha1);
+ if (!cmit)
+ usage(describe_usage);
+
if (!initialized) {
initialized = 1;
for_each_ref(get_name);
```

1. It is preceded with a git diff header, that looks like this (when -c option is used):

```
diff --combined file
```

or like this (when --cc option is used):

```
diff --cc file
```

2. It is followed by one or more extended header lines (this example shows a merge with two parents):

```
index <hash>,<hash>..<hash>
mode <mode>,<mode>..<mode>
new file mode <mode>
deleted file mode <mode>,<mode>
```

The mode <mode>,<mode>..<mode> line appears only if at least one of the <mode> is different from the rest. Extended headers with information about detected contents movement (renames and copying detection) are designed to work with diff of two <tree-ish> and are not used by combined diff format.

3. It is followed by two-line from-file/to-file header

```
--- a/file
+++ b/file
```

Similar to two-line header for traditional *unified* diff format, /dev/null is used to signal

created or deleted files.

4. Chunk header format is modified to prevent people from accidentally feeding it to `patch -p1`. Combined diff format was created for review of merge commit changes, and was not meant for apply. The change is similar to the change in the extended *index* header:

```
@@@ <from-file-range> <from-file-range> <to-file-range> @@@
```

There are (number of parents + 1) @ characters in the chunk header for combined diff format.

Unlike the traditional *unified* diff format, which shows two files A and B with a single column that has - (minus — appears in A but removed in B), + (plus — missing in A but added to B), or (space — unchanged) prefix, this format compares two or more files file1, file2,... with one file X, and shows how X differs from each of fileN. One column for each of fileN is prepended to the output line to note how X's line is different from it.

A - character in the column N means that the line appears in fileN but it does not appear in the result. A + character in the column N means that the line appears in the result, and fileN does not have that line (in other words, the line was added, from the point of view of that parent).

In the above example output, the function signature was changed from both files (hence two - removals from both file1 and file2, plus ++ to mean one line that was added does not appear in either file1 or file2). Also eight other lines are the same from file1 but do not appear in file2 (hence prefixed with +).

When shown by `git diff-tree -c`, it compares the parents of a merge commit with the merge result (i.e. file1..fileN are the parents). When shown by `git diff-files -c`, it compares the two unresolved merge parents with the working tree file (i.e. file1 is stage 2 aka our version, file2 is stage 3 aka their version).

OTHER DIFF FORMATS

The `--summary` option describes newly added, deleted, renamed and copied files. The `--stat` option adds `diffstat(1)` graph to the output. These options can be combined with other options, such as `-p`, and are meant for human consumption.

When showing a change that involves a rename or a copy, `--stat` output formats the pathnames compactly by combining common prefix and suffix of the pathnames. For example, a change that moves `arch/i386/Makefile` to `arch/x86/Makefile` while modifying 4 lines will be shown like this:

```
arch/{i386 => x86}/Makefile | 4 +--
```

The `--numstat` option gives the `diffstat(1)` information but is designed for easier machine consumption. An entry in `--numstat` output looks like this:

```
1 2 README
3 1 arch/{i386 => x86}/Makefile
```

That is, from left to right:

1. the number of added lines;
2. a tab;
3. the number of deleted lines;
4. a tab;
5. pathname (possibly with rename/copy information);
6. a newline.

When `-z` output option is in effect, the output is formatted this way:

```
1 2 README NUL
3 1 NUL arch/i386/Makefile NUL arch/x86/Makefile NUL
```

That is:

1. the number of added lines;

2. a tab;
3. the number of deleted lines;
4. a tab;
5. a NUL (only exists if renamed/copied);
6. pathname in preimage;
7. a NUL (only exists if renamed/copied);
8. pathname in postimage (only exists if renamed/copied);
9. a NUL.

The extra NUL before the preimage path in renamed case is to allow scripts that read the output to tell if the current record being read is a single-path record or a rename/copy record without reading ahead. After reading added and deleted lines, reading up to NUL would yield the pathname, but if that is NUL, the record will show two paths.

GIT

Part of the [git\(1\)](#) suite